COP 5536 Fall 2019

Programming Project

Report of the RisingCity Project

Name: Hsiang-Yuan Liao

UFID: 4353-5341

UF Email: hs.liao@ufl.edu

Project Report

The Use of Data Structure

Basically it uses two data structures: Min Heap and Red Black Tree. For every input data including buildingNum, total_time and executed_time, I store them into a struct node, which both MinHeap and RBT has their pointer pointing at:

```
//Building Data Node which contains all the parameters
we need
struct Node {
    int exec_t;
    int bNum;
    int total_t;
};
```

When ever reads an Insert command from input_file, I create a new Node to store those parameters, and also create new RBT nodes and MinHeap nodes which has a Node pointer point to this new data Node.

```
RBTree Node *ptr  = newNode;
MinHeap Node *ptr = newNode;
```

By doing this, I just need to maintain the data Nodes for executed_time updates, and both data structure could get their reference by accessing the data Node with their pointers.

RBT’s reference is buildingNum => RBT Node->ptr->bNum
MinHeap’s reference is executed_time => MinHeap[i]->ptr->exec_t

The Main Function

Basically it has a main for loop continuing processing the inputs, outputs and building construction algorithm. It will read the first line of the input file before entering the main for loop. We do the following process:

1. If the command global counter = Day counter, execute the command (Insert or Print), and read another line of the command if it is not end of file.
    a. Insert command will create a new Node and connect with Heap/RBT pointer.
    b. Printbuilding(parameter) command will do search in RBT, if it is found, it will print (bNum, exec_time, total_time), if it is not found print (0,0,0) instead.
    c. Printbuilding(para1, para2) command will do a special search that returns the first RBT node that it’s bNum is set between para1 and para2, which is para1<= RBTNode->ptr->bNum <= para2. And I use this node as root, to do a Inorder Traversal Print, inorderly print the building who’s bNum is in that range. By using this

   method, it will be faster than using a for loop with pure RBT search.

2. Pick a target building for construction. Get the root of the min heap as the target building, it will continually construct until it's exec_t equals total_t or it reaches the MAX_PERIOD = 5 of construction, whichever happens first.
    a. If it's exec_t equals total_t, print it out and remove it from RBT before delete the data node.
    b. If the construction period is up to time, insert it back to MinHeap, before choosing for another target building.
3. If target building is not NULL, then let the exec_t + 1 for every day.

The Function Prototype

```
class RBTNode{
     RBTNode(void) //constructor
     bool isLeftChild(void) // check the node is the left child
of its parent
     bool hasRedChild(void) //check if the node has at least one
red child
     RBTNode *sibling(void) //get the sibling of the RBTNode
     RBTNode *PP_sibling(void) //get the parent's sibling
}

class RBTree {
     void leftRotate(RBTNode *current) //do left rotation at
that RBTNode
```

```
     void rightRotate(RBTNode *current) //do right rotation at
that RBTNode
     void colorSwap(RBTNode *x, RBTNode *y) //swap the color of
two RBTNode
     void nodeSwap(RBTNode *x, RBTNode *y) //swap the pointer of
two RBTNode
     void insert_fix(RBTNode *current) //fix RBTree after insert
     RBTNode *rightmostnode(RBTNode *x) //get the replaced node
for BST delete
     void deleteNode(RBTNode *delNode) //delete the given node
in the tree
     void rebalanceRBT(RBTNode *x) //rebalance the RBTreee after
deletion
public:
     RBTNode *search(int bn) //search for the node by reference
(building number)
     Node *printSearch(int bn)
          //if it is found, return the data node we are looking
          for, if it is not found, return NULL
     RBTNode *inorderRootSearch(int bn1, int bn2)
          //if it is found, return the first RBTree node, that
          has the bNum which is bn1 <= bNum <= bn2, if it is not
          found, return NULL

     void inorderPrint(RBTNode *root, int bn1, int bn2, FILE
*fout, int *print_comma)
          //inorder print all the nodes which bNum is between
bn1 and bn2
     void rbtinsert(Node *N) //insert the node in RBTree
     void rbtdelete(int bn) //delete the node in RBTree
}

class MinHeap {
     MinHeap(int n) //constructor
     void swap(MinHeapNode *x, MinHeapNode *y)
          //swap the MinHeapNode pointer to the data node
     int parent(int i)  //get the index of parent/left child/
right child
     int left(int i) //get the index of parent/left child/right
child
     int right(int i) //get the index of parent/left child/right
child
     void hinsert(Node *newNode)
          //Let the min heap node ptr point to the new data node
for heap insertion
     Node *removeMin(void)
          //remove the root from min heap and get the data node
          that root pointer is pointing at
     void heapify(int i)
          //heapify function, the reference is the exec_time
          that the heap node pointer is pointing at
     Node* getMin(void) //get the root node ptr
}

int readInput(FILE * fp, int *curInputID, int *parameter1, int
*parameter2)
     //Read Input Line by Line and return the Input command type
Node *CreateNewNode(int para1, int para2)
     //execute command function, handle three kind of commands
int main(int argc, char** argv) //Main function for rising city
```